LSV Final Project: Linear Reversible Circuit Synthesis and Optimization

Yen-Hsiang Huang
Taipie, Taiwan
Email: b06901170@ntu.edu.tw

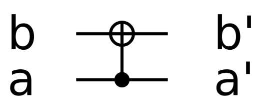
***Abstract*—In the final project of this class, I choose to do a survey-type and re-implementation project. The survey domain is linear reversible circuit synthesis and optimization, and it is especially for CNOT gate circuit(Thus, it is a linear reversible circuit).**

To synthesize a CNOT circuit based on Gaussian elimination, given the output functions of a linear reversible circuit, we can easily obtain a CNOT circuit by recording each row operation done. And the idea of optimize a circuit is based on the idea of synthesis. Given a circuit consists of CNOT gate, by traverse over the CNOT gates, we can easily obtain the output function. With the output function, we can re-synthesis the circuit again and we would potential get a circuit with less gates.

Keywords: CNOT gate, linear reversible circuit, circuit synthesis, circuit optimization

I. Introduction

A CNOT gate is a quantum logic gate that consists of two quantum bits(qubits), one is called control and the other is called target. When control bit is $|1\rangle$, target bit will flip, while the control remain the same(see Fig.1b). Thus, target bit acts like a exclusive-or gate in ordinary circuit, or, a bitwise addition.



(a) a CNOT gate

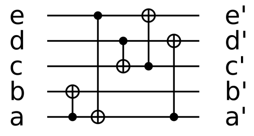
a	b	a'	b'
1	0	1	1
1	1	1	0
0	0	0	0
0	1	0	1

(b) true table of CNOT

Fig. 1: a CNOT gate with a,b being the inputs, and a', b' being the outputs

When operating standard Gaussian elimination upon a matrix, all the steps are row operations, and our goal is use row operations to make the matrix become a identity matrix. Row operations and CNOT have similar behavior, they both reserve one row(or qubit), and add it to another. If the matrix is a binary matrix, and the addition between two rows is bitwise addition(exclusive-OR), then the behavior is even more similar. Thus, we can synthesize circuit by using Gaussian elimination if we could know the output functions, and the output functions is reversible. Below, the standard method and its reason of synthesis a CNOT circuit will be explained.

First, given the output functions, we can generate a output matrix by treating every output as a row and every input as a column(see Fig.2b). Then, if we append any new CNOT gate, for example, a CNOT gate with a' being the control and b' being the target, the resulting b" will become $a+e+a+b = b+e$, and it can be obtained by adding the row a' to the row b'($R_5 \rightarrow R_4$) in the matrix.



(a) a CNOT circuit

	e	d	c	b	a	
e'	1	1	1	0	0	c+d+e
d'	1	1	0	0	1	a+d+e
c'	0	1	1	0	0	c+d
b'	0	0	0	1	1	a+b
a'	1	0	0	0	1	a+e

(b) matrix represent the outputs of 2a

Fig. 2: an example CNOT circuit and its output matrix

Therefore, if we can derive a $n \times n$ identity matrix from the matrix through a sequence of row operations, which is what Gaussian elimination's goal, it means that we can derive the original input from the output by appending a corresponding sequence of CNOT gate. Obviously, this sequence of CNOT gates is a realization of a circuit, whose input is the output of the original circuit, and vice versa.

A CNOT circuit and another circuit with totally reversely sequence of CNOT gates are reverse circuit to each other. This can be proven by appending one to the other, and here is a example(see Fig.3). If we can prove that $a = a', b = b', ..., e = e'$, than we can assert that $C1$, $C2$ are reverse circuit to each other. It is obvious that we can remove the two identical CNOT gates in the middle, which are the gates with the fifth line being the control and the second line being the target in this case, without affecting the functionality of the whole circuit. Then, we can remove the two identical gates again(control: third line, target: first line). By doing this iteratively, the gates will be totally removed eventually, thus, $a = a', b = b', ..., e = e'$, and $C1$, $C2$ are reverse circuits.

Hence, if we can derive a $n \times n$ identity matrix by Gaussian elimination from the matrix of the circuit, than we can obtain a realization of the given output functions. Note that if a identity matrix cannot be derived, it means that the rows of matrix is not linear independent, and the circuit is not a reversible circuit because of the lost of information.

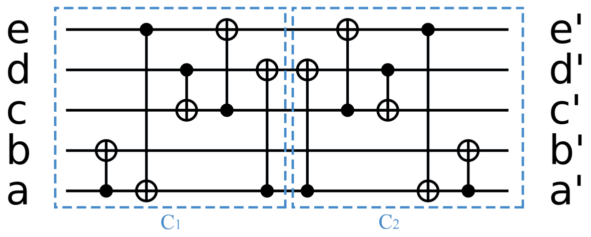


Fig. 3: a CNOT circuit appended by its inverse circuit

Some previous works have been done in this domain. I have studied the following two papers [3][4], which are in different scenarios, re-implemented the algorithm proposed in them and did some experiments.

The remainder of this report is as follows. In Sec.II and Sec.III A brief introduction to the algorithm will be performed, with some additional explanation about something I consider important but does not appear in [3][4]. In Sec. IV I will briefly explain the method I used in my re-implementation. In Sec.V the results will be shown. In Sec.VI I will discuss the observation, and propose some future works.

II. Efficient Synthesis[4]

In [4], Ketan N. Patel, Igor L. Markov and John P. Hayes[4] have provided an optimal method to synthesize a CNOT circuit in terms of the complexity of gates. Although synthesizing through Gaussian elimination is an simple and intuitive way, its complexity is a problem.

The row operation's complexity of Gaussian elimination is $O(n^2)$, because the i^{th} column needs $O(1)$ row operation to make the i^{th} entry become 1, and $O(n)$ row operations to eliminate all the 1's except the i^{th} entry. However, a suggestion is given that the lower bound of row operations is $\Omega(\frac{n^2}{\log(n)})$[4, p.4]. Hence, there might be a more efficient method than Gaussian elimination. Note that the complexity mentioned above is the complexity of CNOT gates. A row operation's time complexity is $O(n)$, thus the time complexity of Gaussian elimination is $O(n^3)$, and the lower bound is hence $\Omega(\frac{n^3}{\log(n)})$[4].

An algorithm Lwr_CNOT_Synth with complexity being $O(\frac{n^2}{\log(n)})$ for making a matrix become upper triangular form through row operation had been proposed in [4].

First, Lwr_CNOT_Synth use a divide-and-conquer technic by partition the matrix into several regions all consist of $< m$ successive columns, which are called $column\ sections$, and the entries in a $column\ section$ within the same row are called $sub-rows$[4, p.5].

Second, Lwr_CNOT_Synth has the following three crucial steps: Consider a $column\ section$ having m columns, beginning with the i^{th} column in a $n \times n$ matrix.

Step A For the j^{th} $sub-row$, $i \leq j \leq n$, record the first appearance for each possible pattern. Note that there might be at most 2^m patterns. Then, for each pattern, use the corresponding recorded $sub-rows$ to eliminate the duplicate $sub-rows$[4, p.5].

Step B For the j^{th} row R_j, $i \leq j \leq \min\{n, i+m-1\}$, check whether the j^{th} entry is 1 or not. If not, pick the closest row R_k, $j < k \leq \min\{n, i+m-1\}$, with the j^{th} entry being 1, and apply row operation $R_k \rightarrow R_j$ to "diagonalize" the $column\ section$[4, p.5].

Step C For the j^{th} column C_j, $i \leq j \leq \min\{n, i+m-1\}$, check whether the entries e_k, $j \leq k \leq n$ remains 1 or not. If it does, apply $R_i \rightarrow R_k$ to eliminate the remaining 1[4, p.5].

Lwr_CNOT_Synth is applied in this way:

Step 1 Use Lwr_CNOT_Synth to obtain a upper-triangular matrix[4, p.5].

Step 2 Transpose the resulting upper-triangular matrix, thus, the matrix become lower-triangular[4, p.5].

Step 3 Use Lwr_CNOT_Synth again on the resulting lower-triangular matrix, and it will become a $n \times n$ identity matrix[4, p.5].

Step 3 is feasible because the any row operation $R_i \rightarrow R_j$ done by the algorithm follows the inequation, $i < j$, meaning that it guarantee the matrix being lower-triangular during Step 3.

By recording the row operations in Step 1, we can obtain a circuit called $circuit_l$. Similarly, a circuit called $circuit_u$ can be obtained from Step 3[4, p.6]. [4] says that the complete circuit should first switch the control/target of CNOT gates in $circuit_u$, reverse $circuit_u$, and append $circuit_l$ in the back of $circuit_u$. The reason is explained below:

Every row operation $R_i \rightarrow R_j$ applied on a matrix can be seen as a multiplication by an elementary matrix $T_{(j,i)}(1)$[4, p.3]. And, binary elementary matrix $T_{(j,i)}(1)$ has a special property: the inverse matrix of $T_{(j,i)}(1)$ is itself.

Suppose that a matrix A is reduced to matrix B in Step 1 by a sequence of row operation $\{T_l\}$, and B^T is reduce to a identity matrix in Step 3 by $\{T_u\}$. That is,

$$\prod_{a=1}^{n} T_{l_a,(i_a,j_a)} A = B \tag{1}$$

$$\prod_{b=1}^{m} T_{u_b,(k_b,l_b)} B^T = I \tag{2}$$

Here, sequence$\prod a = 1^n T_{l_a,(i_a,j_a)}$ represents $circuit_l$, and $\prod_{b=1}^m T_{u_b,(k_b,l_b}$ represent $circuit_u$. From 2 we can obtain 3 by applying transpose on both side.

$$BT_{u_m,(k_m,l_m)}T_{u_{m-1},(k_{m-1},l_{m-1})}...T_{u_1,(k_1,l_1)} = I \tag{3}$$

Combine 1 and 3, we can obtain

$$(\prod_{a=1}^{n} T_{l_a,(i_a,j_a)})A(\prod_{b=0}^{m-1} T_{u_{m-b},(k_{m-b},l_{m-b})}) = I \tag{4}$$

And thus,

$$A = \prod_{a=0}^{n-1} T_{l_{n-a},(i_{n-a},j_{n-a})} \prod_{b=1}^{m} T_{u_b,(k_b,l_b)} \tag{5}$$

As mentioned in Sec.I, $\prod_{a=0}^{n-1} T_{l_{n-a},(i_{n-a},j_{n-a})} \prod_{b=1}^{m} T_{u_b,(k_b,l_b)}$ represent the reverse circuit of A, Therefore, the circuit that can realize A will be its reverse sequence, that is , $\prod_{b=0}^{m-1} T_{u_{m-b},(k_{m-b},l_{m-b})} \prod_{a=1}^{n} T_{l_a,(i_a,j_a)}$. Obviously, it can be obtain by the operations of the $circuit_l$ and $circuit_u$ mentioned above.

III. Efficient Synthesis with connectivity constraints[3]

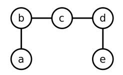
Considering the connectivity of the qubits in the physical design, it might not be possible for any two qubits interacting with each other. To perform a CNOT operation $CNOT(i,j)$(meaning that q_i being the control and q_j being the target) between any two qubits q_i, q_j that are not neighbors without affecting the other qubits, a template can be perform[3]:

Step 1 Find a path $\{q_p\}$ from q_i to q_j in the connectivity graph.

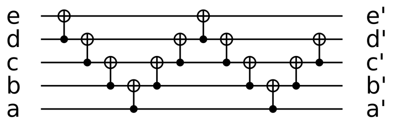
Step 2 Apply the CNOT gates in the four sequences explained below:

$$\begin{aligned}&CNOT(p_n,j)CNOT(p_{n-1},p_n)...CNOT(i,p_1)\\&CNOT(p_1,p_2)CNOT(p_2,p_3)...CNOT(p_n,j)\\&CNOT(p_{n-1},p_n)CNOT(p_{n-2},p_{n-1})...CNOT(i,p_1)\\&CNOT(p_1,p_2)CNOT(p_2,p_3)...CNOT(p_{n-1},n)\end{aligned} \tag{6}$$

The first and second sequence is to make the qubits in $\{q_p\} \cup q_j$ become $(q_p + q_i)$ or $(q_j + q_i)$, while the third and forth sequence makes the qubits in the path resume to $\{q_p\}$ again. An example of how to apply $CNOT(a,e)$ on qubits a and e is shown in 4. It is obvious that a template like this cost $O(n)$ gates.



(a) connectivity graph



(b) the template

Fig. 4: a template that can perform CNOT to qubit a and e. Adoted from [3, p.2]

Thus, the method mention in Sec.II become $O(\frac{n^3}{\log(n)})$[4, p.2] considering the complexity of CNOT gates. In [4], Beatrice Nash, Vlad Gheorghiu, and Michele Mosca[3] proposed a new method, based on [4], cost only $O(n^2)$ with the connectivity constraints. First, they didn't use the partitioning idea in [4], complaining that this idea works not that good at sparse connectivity[3, p.2-3]. Then, a new Lwr_CNOT_Synth is proposed. Because the partitioning step in [4] is neglected, we can neglect Step A in [4]. The adapted Step B , Step C is as follow[3, p.4]:

Considering the i^{th} column C_i in the matrix,

Step B If the i^{th} entry e_i in C_i is 0, find all the entry $\{e_j\}$ such that $e_j = 1,\ j > i$. For all e_j, find the one such that q_j is closest to q_i in the connectivity graph, and perform the template mentioned in (6).

Step C Collect all the entry $\{e_j\}$ such that $e_j = 1,\ j \geq i$. Second, construct a Steiner tree T_i with $\{q_j\}$ in the given connectivity graph. Note that q_i is called the $root$ and the qubits in $\{q_j\}$ are called $terminals$, while the remainder qubits in T_i is called $Steiner\ nodes$[3, p.3]. Third, traverse T_i using BFS from its $root$ to construct the $sub-trees$. A $sub-tree$ should have a $terminal$ being its $root$ and leaves, while the remainder of it only consists of $Steiner\ nodes$. So, a $sub-tree$'s $root$ is another $sub-tree$'s leaf or T_i's $root$. Then, from the last $sub-tree$ constructed, applied the template mentioned in (6) for its $root$ and all of its leaves. Note that the shared ancestor nodes of two leaves in a $sub-tree$ can be transform to corresponding shared CNOT gates in the template. The sharing method is done traverse T_i through a reverse DFS order when applying the template. Thus it cost $O(n)$ gates in Step C.

In Step C, the main idea is to first use the corresponding entry of a $sub-tree$'s $root$ to eliminate the one of its leaves. Then, use the entry of the ancestor $sub-tree$'s $root$ to eliminate the one of the descendant $sub-tree$'s $root$. By doing this iteratively, the entries of T_i's $terminals$ will be eliminated eventually.

However, this method would face a problem after the matrix has been transposed, because it done not guarantee that the entry of a $sub-tree$'s $root$ and leaves e_r and $\{e_l\}$ satisfy the inequation, $r < l$. Thus, the lower-triangular form may be ruined[3, p.4].

Therefore, a exclusive edition of Step C on a transposed matrix are needed. There is an example in [3, FIG.6], however, there is no explanation in detail:

Step C Collect all the entry $\{e_j\}$ such that $e_j = 1,\ j \geq i$. Second, construct a Steiner tree T_i with $\{q_j\}$ in the given connectivity graph. Instead of constructing $sub-trees$ and applied the template mentioned in (6) on the $sub-trees$, applied the template directly on T_i. However, the third and forth sequences of (6) should be adapted.

If a qubit q_c is a terminal but is neither the $root$ nor the leaves of T_i, and suppose that the first CNOT gate with q_c being the control is $CNOT(c,t)$ in the thrid sequence, the last CNOT gate with q_c being the control in the forth sequence should also be $CNOT(c,t)$. Change these two $CNOT(c,t)$ to $\{CNOT(c,t)CNOT(t,c)CNOT(c,t)\}$. The role of $\{CNOT(c,t)CNOT(t,c)CNOT(c,t)\}$ is swapping the signals of q_t and q_c. And after doing this, the signal of q_c is remained unchanged before and after the third and forth sequence, instead of resuming to the original signal.

Now, we have the algorithm Lwr_CNOT_Synth on a matrix after or before transposed. Applied the method in [4], and we can obtain the circuit with connectivity constraints. Since Step C's cost $O(n)$ gates, the total gate complexity become $O(n^2)$.

IV. Re-implementation

A re-implementation has been attached. Below is the environment:

OS	Ubuntu20.04
CPU	AMD R5 3600@3.6GHz
Memory	16Gb@3200MHz
Language	C++17

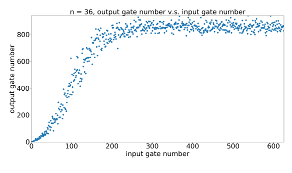
The result of re-implementation of [4] is given in the attached folder LSV_Final. In [4], the authors mentioned that the CNOT gate complexity of the whole algorithm is $O(\frac{n^2}{\alpha \log_2 n} + 2n^{1+\alpha})$[4, Equation5], where the column-sections' width m is no more than $\alpha \log_2 n$. To obtain a complexity of $O(\frac{n^2}{\log(n)})$, I chose α to be $\frac{1}{2}$. Thus, the total gate complexity become $O(2\frac{n^2}{\log_2 n}) = O(\frac{n^2}{\log(n)})$.

In Step C of Lwr_CNOT_Synth in [3], a approximated Steiner tree is generated by BFS from the $terminals$, and once two $terminals$' paths collide together, the paths will be combined into a super-node, and been treated as another $terminal$. By interatively doing this, the $terminals$ will eventually been combined into a single node, and the process is done[3, p.4].

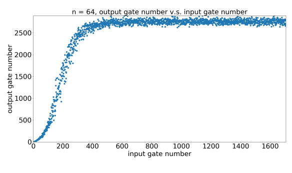
During the re-implementation of [3], I found that the connectivity graph, such as Google Bristlecone 72 qubit architecture, the IBM Tokyo 20 qubit architecture, and the Rigetti's Acorn 19 qubit architecture[3, p.6] are all semi-grid graph. If a connectivity graph is grid-based, it implies that we can use some more efficient and accurate waies to generate a Steiner tree, such as FLUTE[1]. Moreover, [2, p.208-209] suggest that a Steiner tree problem in graph remains NP-complete if all edges weights are equal, and a geometric Steiner tree problem also remains NP-complete if the distance measure is rectilinear metric. Thus, there might be some transformation that can reduce the Steiner tree problem on the semi-grid graph(the problem here, and the edge weights are all equal) to a rectilinear Steiner tree problem(what can be solved by FLUTE[1]). Unfortunately, a efficient way of transformation of these two problem has not been done yet, thus, the re-implementation here is just for grid-based connectivity graph.

V. Results

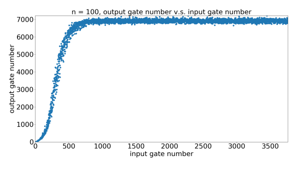
Results of the implementation of [4] and [3] is in Fig.5 and Fig.6, respectively.



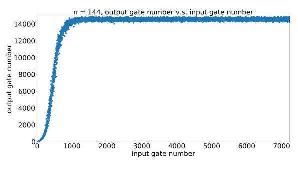
(a) #qubits = 36



(b) #qubits = 64

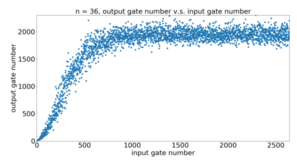


(c) #qubits = 100

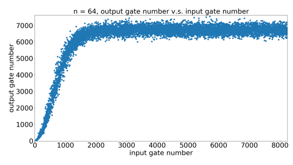


(d) #qubits = 144

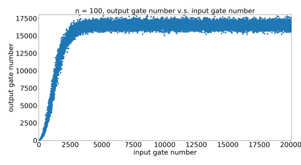
Fig. 5: an example CNOT circuit and its output matrix



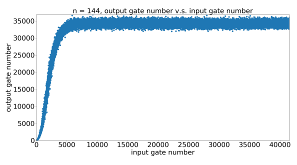
(a) #qubits = 36



(b) #qubits = 64



(c) #qubits = 100



(d) #qubits = 144

Fig. 6: an example CNOT circuit and its output matrix

The x-axis represent the gates number of the input circuit, and the y-axis represent the gates number of the output circuit. The input circuit is generated by randomly generate CNOT gates, however, the generation also consider connectivity constraints if the connectivity graph is given. By observation, the resulting output circuit size increase dramatically when the input circuit size increase at the beginning. However, it will be convergent when input circuit size getting larger. We can plot the convergent output circuit size against the number of qubits(see Fig.7 for the result without connectivity constraints and Fig.8 with connectivity constraints)

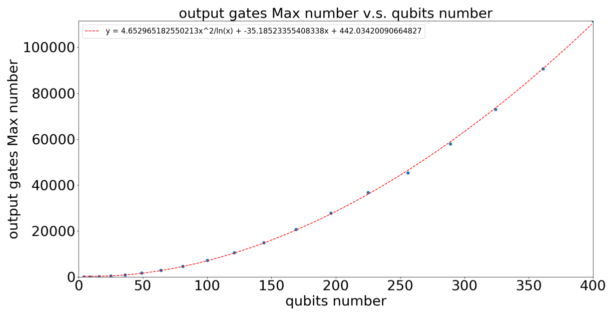


Fig. 7: convergent output circuit size v.s. # qubits (without connectivity constraints

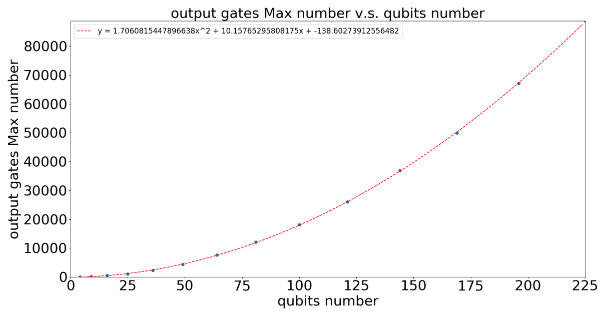


Fig. 8: convergent output circuit size v.s. # qubits (with connectivity constraints

The resulting regression function is $y = 4.653\frac{x^2}{\ln x} - 35.2x + 442$ for Fig.7 and $y = 1.7x^2 - 10x - 138.6$ for Fig.8. It can be seen as a loose verification for the gates complexity calculated in [4] and [3].

VI. Discussion

Although these two method are both efficient in different scenarios, there are something that can be improved. Both of these methods, although they are applied in different scenarios, have the same issue: the output circuit size increase dramatically in small input circuit size. The possible reason for this phenomenon is that when focusing at a column(or a column-sections), the information of the other columns(or column sections) is neglected. This might make the unsolved columns(or column-sections) getting more and more $1's$ during the process, and as the $1's$ increase, the output circuit size increase. To solve this, we might need to take the information of the unsolved columns(or column-sections) into consideration during the process.

In [3], the authors suggest to not use partitioning, due to the poor performance. This phenomenon is because of the increase number of Steiner trees. Consider a column-section with width m, $O(2^m)$ Step As, and $O(m)$ Step Cs are needed, thus $O(2^m)$ Steiner trees are constructed. However, without partitioning, only $O(m)$ Steiner trees are needed.

In the future, there might be two major things that can be studied. First, is to find a efficient method to reduce a Steiner tree problem on a semi-grid graph to a rectilinear Steiner tree problem, which mention in Sec.IV. Second, is to decide how to place the qubits on a given connectivity graph architecture, with a given output matrix.

References

[1] C. Chu and Y. Wong. “FLUTE: Fast Lookup Table Based Rectilinear Steiner Minimal Tree Algorithm for VLSI Design”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 27.1 (2008), pp. 70–83. DOI: 10.1109/TCAD.2007.907068.

[2] Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. USA: W. H. Freeman Co., 1990. ISBN: 0716710455.

[3] Beatrice Nash, Vlad Gheorghiu, and Michele Mosca. “Quantum circuit optimizations for NISQ architectures”. In: *Quantum Science and Technology* 5.2 (2020), p. 025010.

[4] Ketan N Patel, Igor L Markov, and John P Hayes. “Efficient synthesis of linear reversible circuits”. In: *arXiv preprint quant-ph/0302002* (2003).